



ELB-trees - Efficient Lock-free B+trees

Bonnichsen, Lars Frydendal; Karlsson, Sven ; Probst, Christian W.

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

Citation (APA):
Bonnichsen, L. F., Karlsson, S., & Probst, C. W. (2013). *ELB-trees - Efficient Lock-free B+trees*. Poster session presented at 9th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems, ACACES 2013, Fiuggi, Italy.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

ELB-trees - Efficient Lock-free B+trees

Lars Frydendal Bonnichsen, Sven Karlsson, and Christian W. Probst

Technical University of Denmark



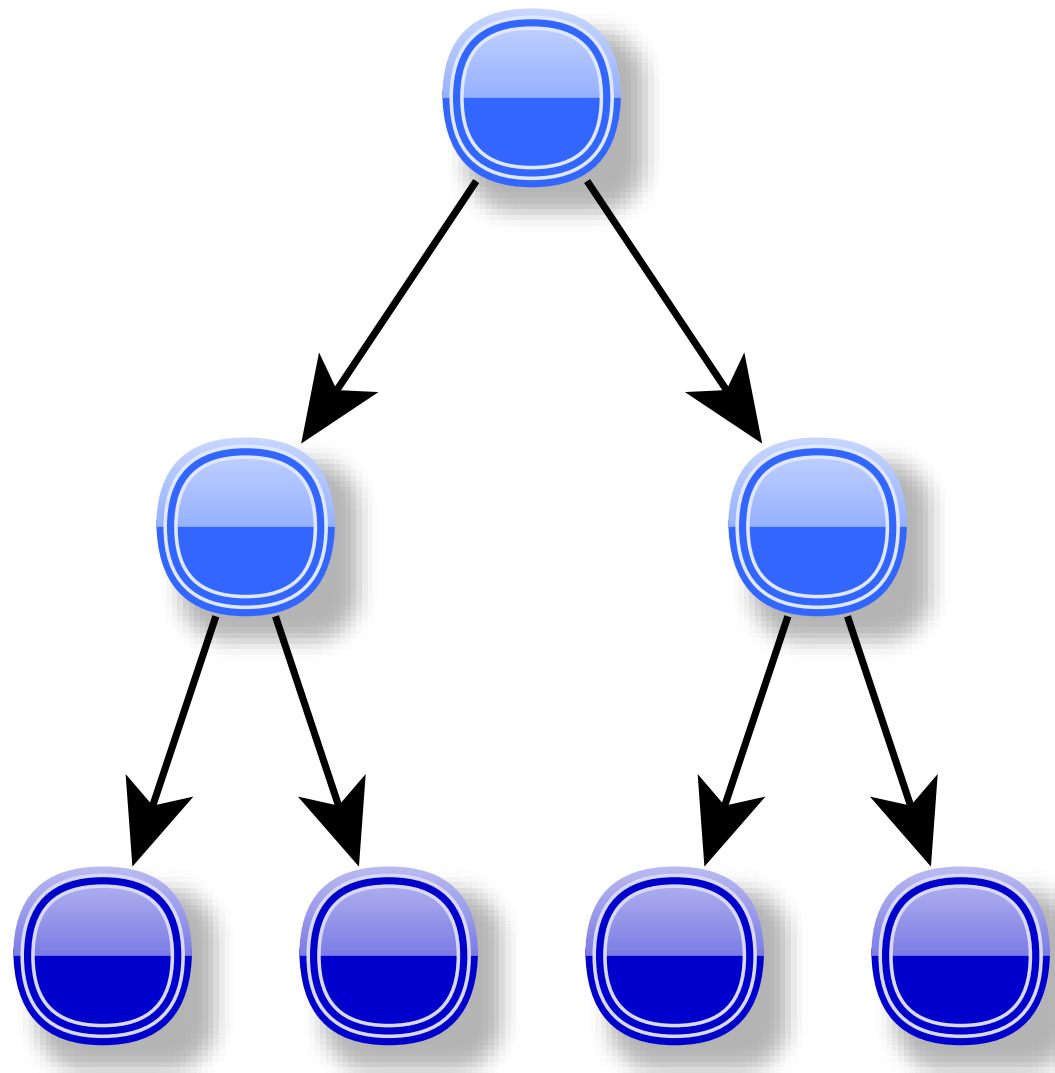
Motivation

- ▶ Processors are increasingly parallel
- ▶ We need scalable, efficient, and thread safe data structures
- ▶ Lock based solutions scale poorly
- ▶ Lock-free solutions avoid deadlocks and scheduling issues

Contributions

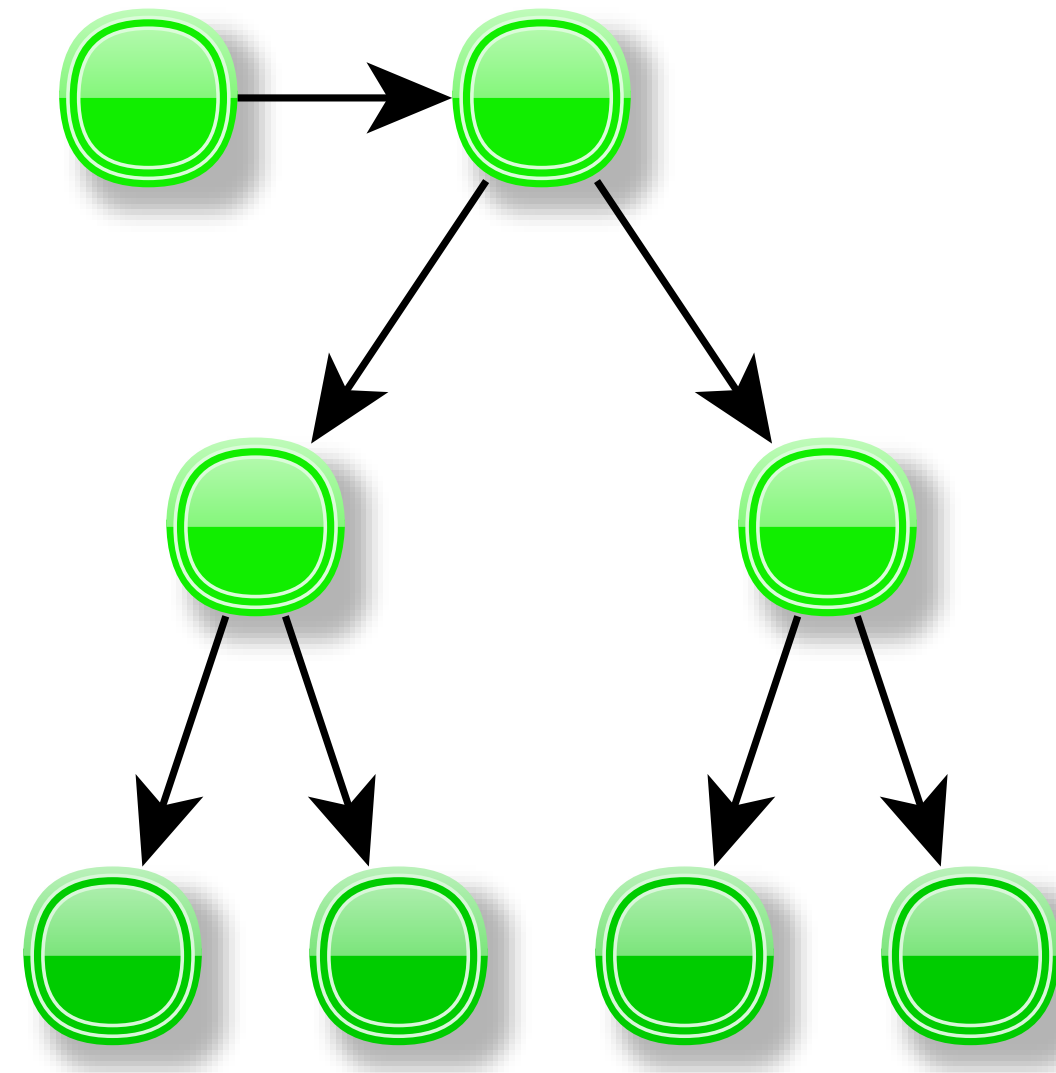
- ▶ An efficient lock-free balanced search tree (ELB-tree)
- ▶ Uses a single synchronization (CAS) per operation, in the common case
- ▶ Not dependent on reference counting or automatic garbage collection
- ▶ Almost 30 times faster than left-leaning red-black trees at 30 threads

B+trees (Inspiration)



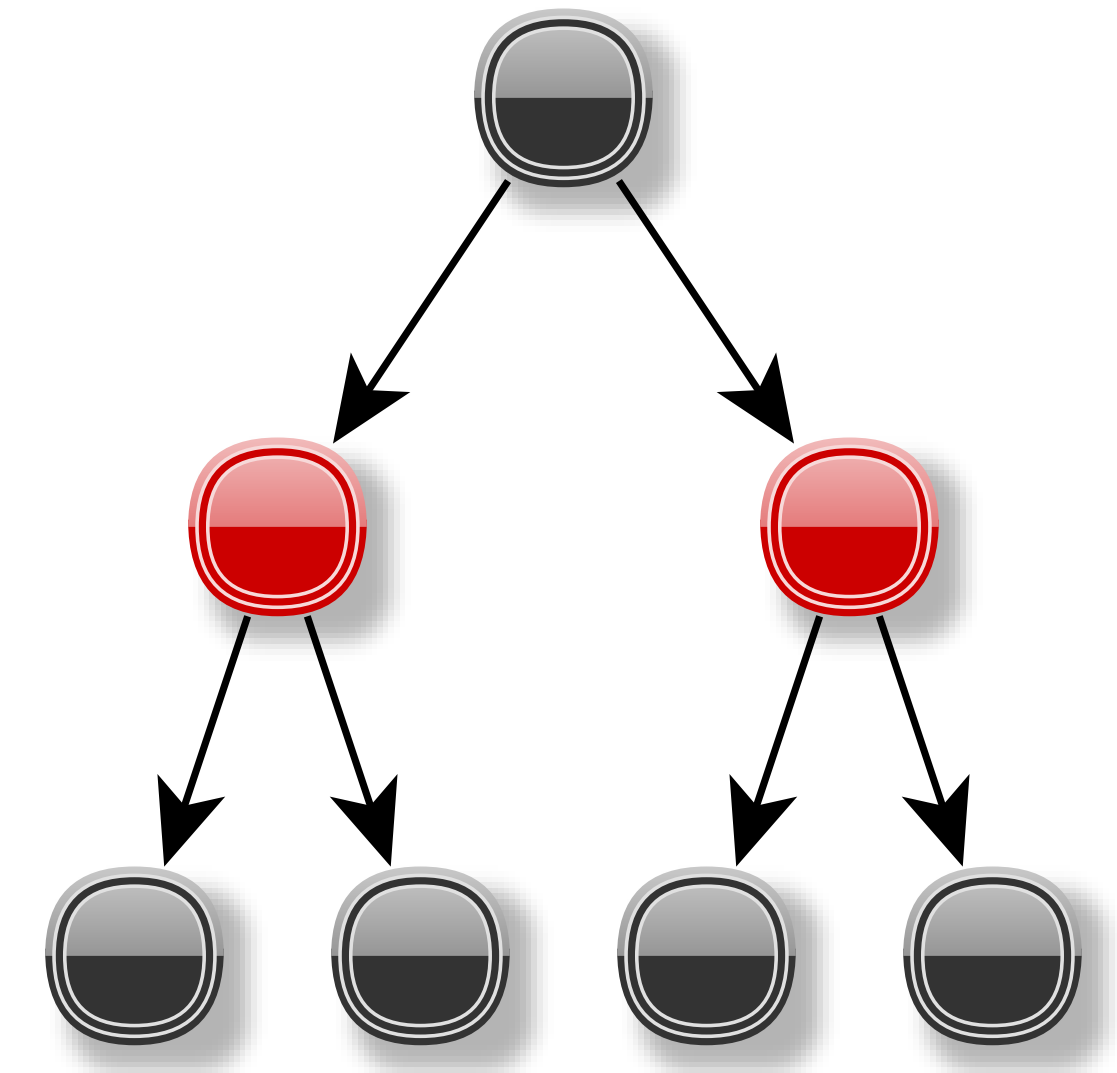
- ▶ Leaf oriented wide search tree
- ▶ Nodes at least 50% full
- ▶ Rebalance by merging, splitting, or stealing
- ▶ Optimized for space and storage on media

ELB-trees (New)



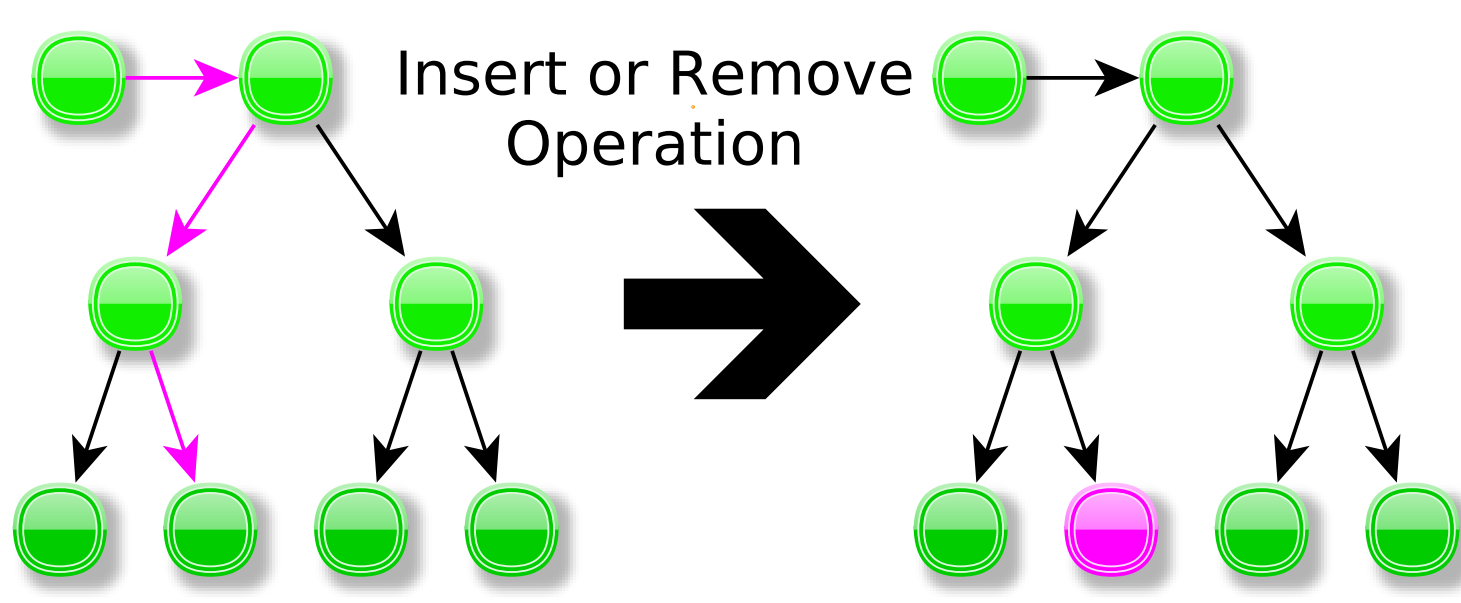
- ▶ Leaf oriented wide search tree with fake root
- ▶ Nodes at least k^{-1} , $k > 2$ full
- ▶ Rebalance by replacing parent node
- ▶ Optimized for parallel speed and RAM storage

Left-leaning red-black trees

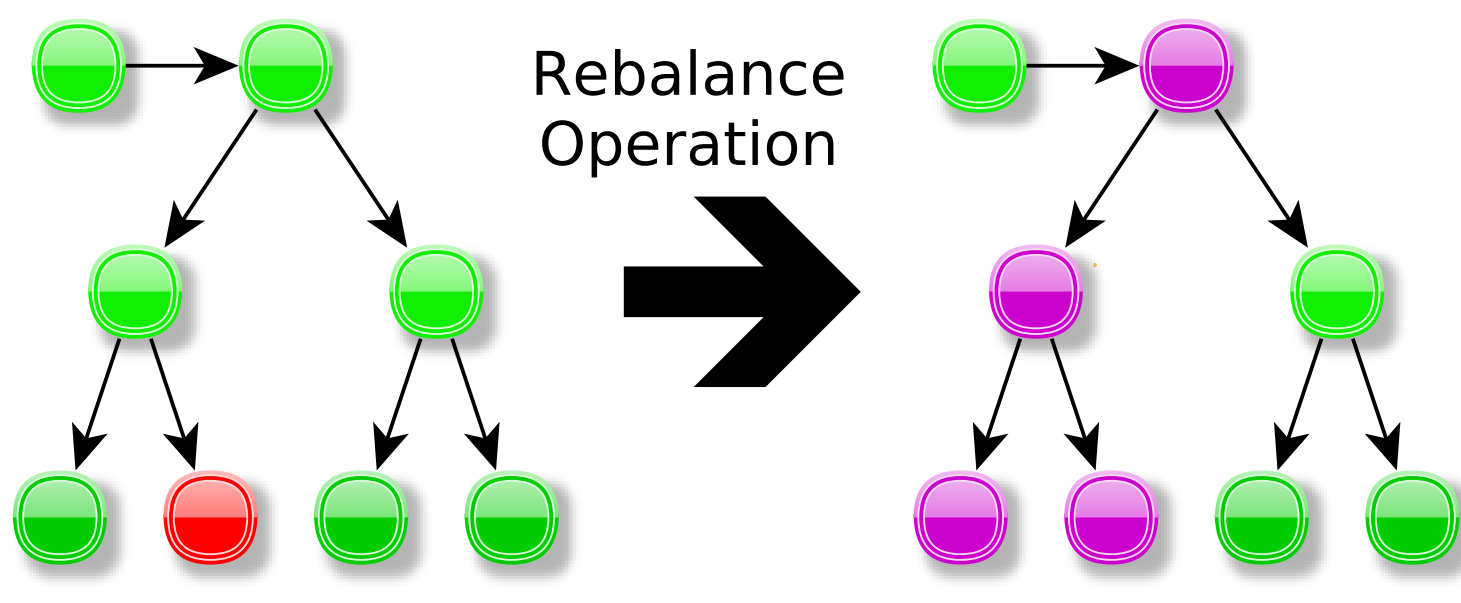


- ▶ State of the art binary search tree
- ▶ No empty nodes
- ▶ Local rebalance frequently
- ▶ Optimized for speed and RAM storage

Approach



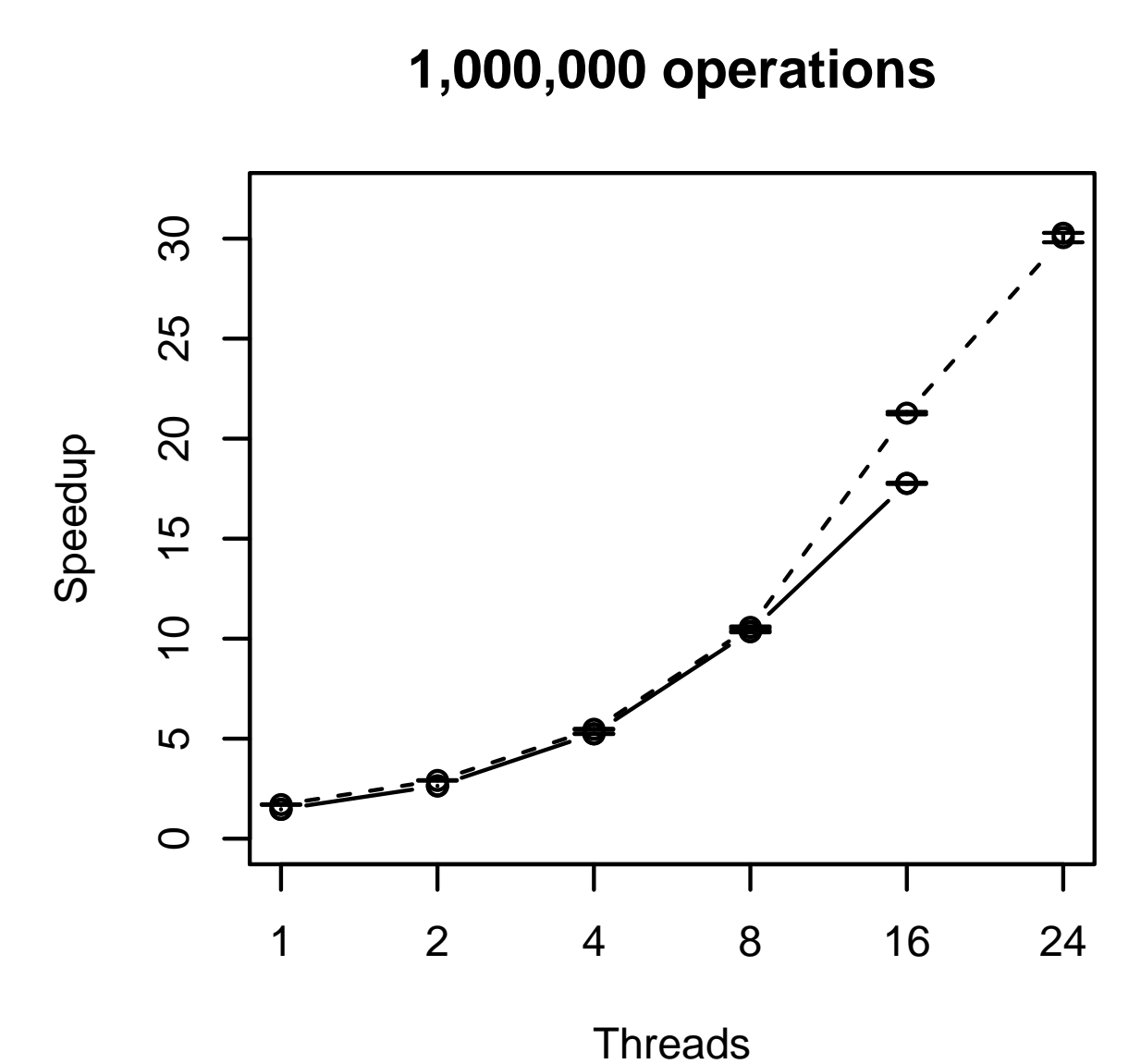
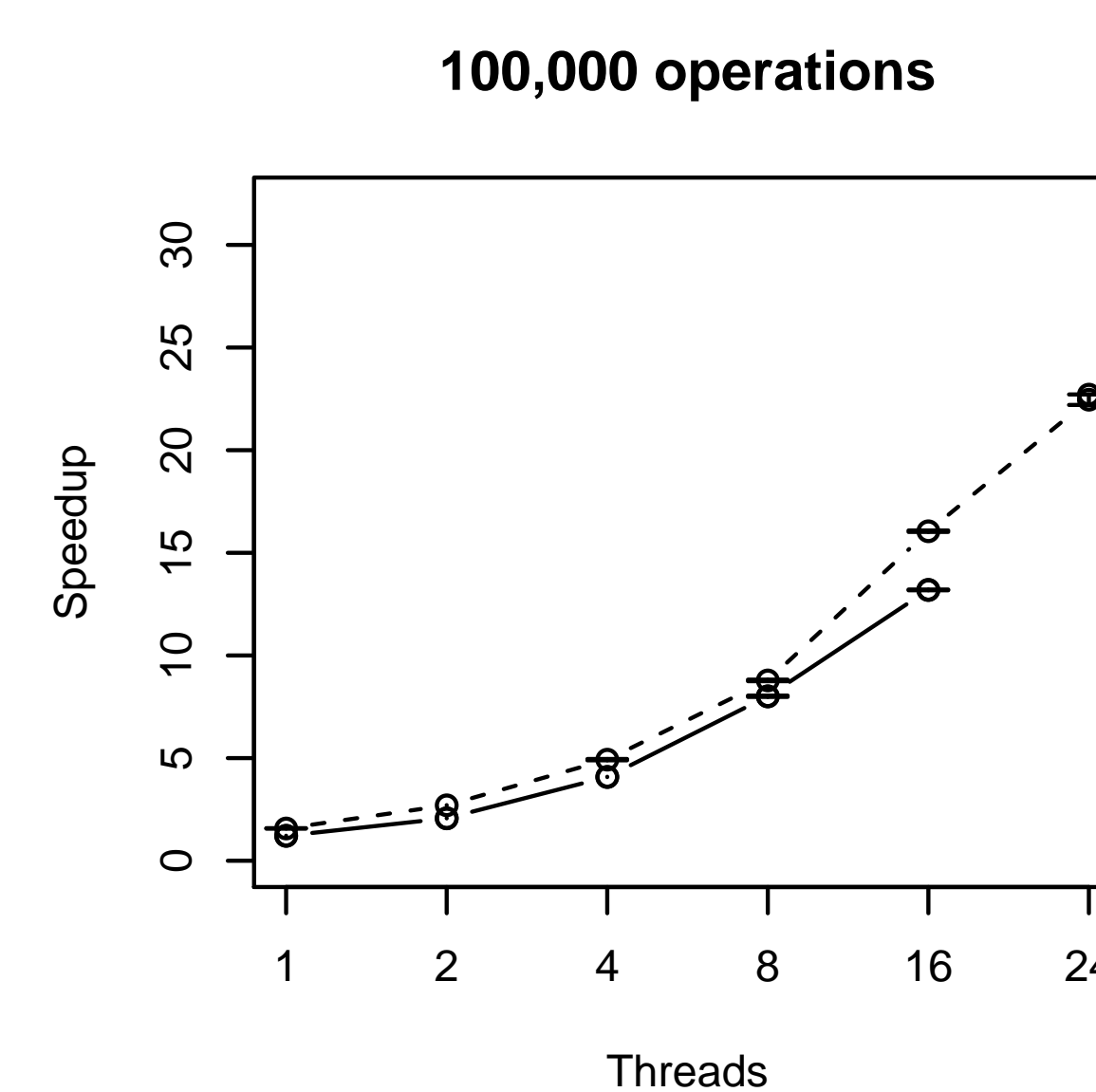
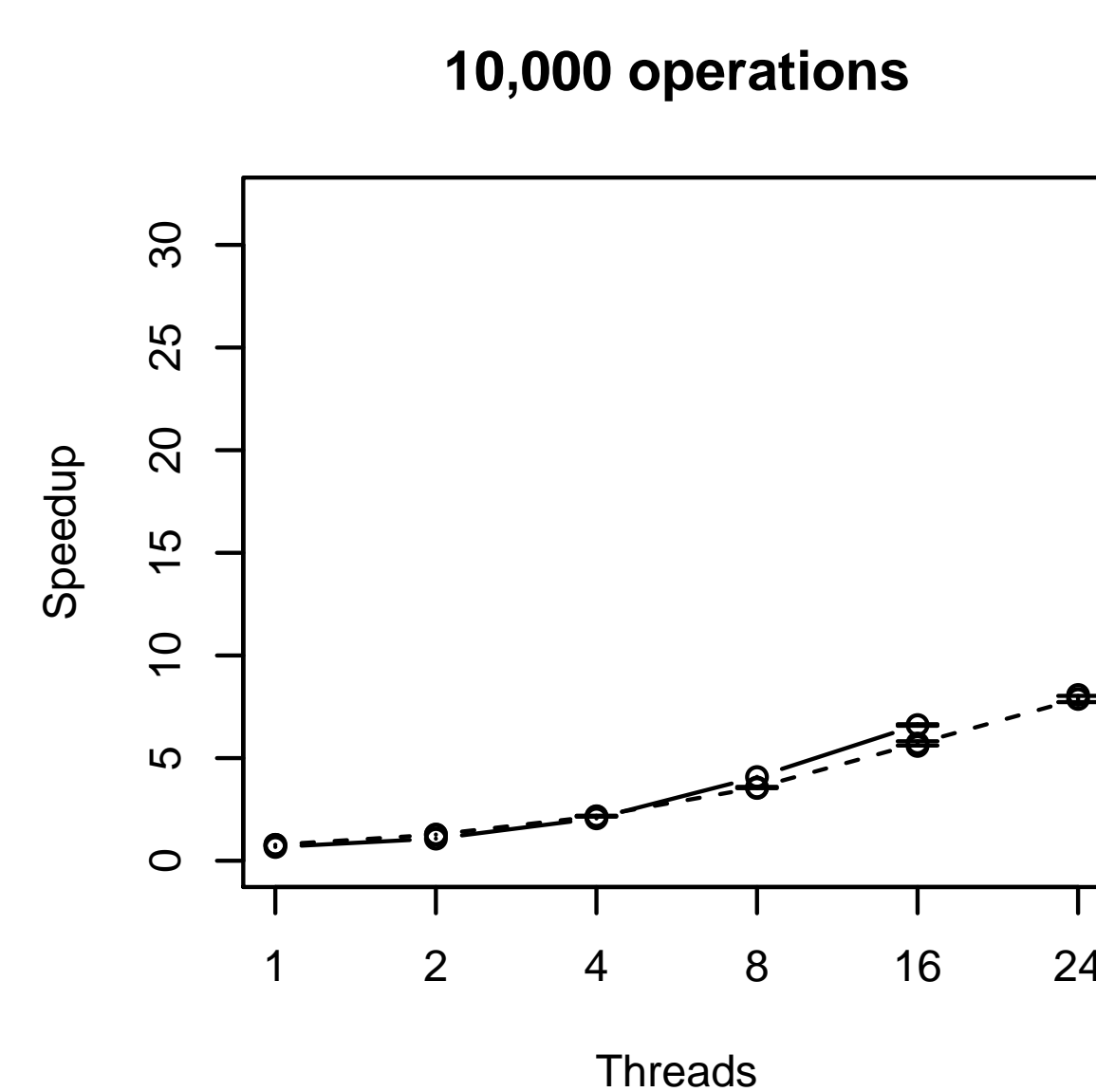
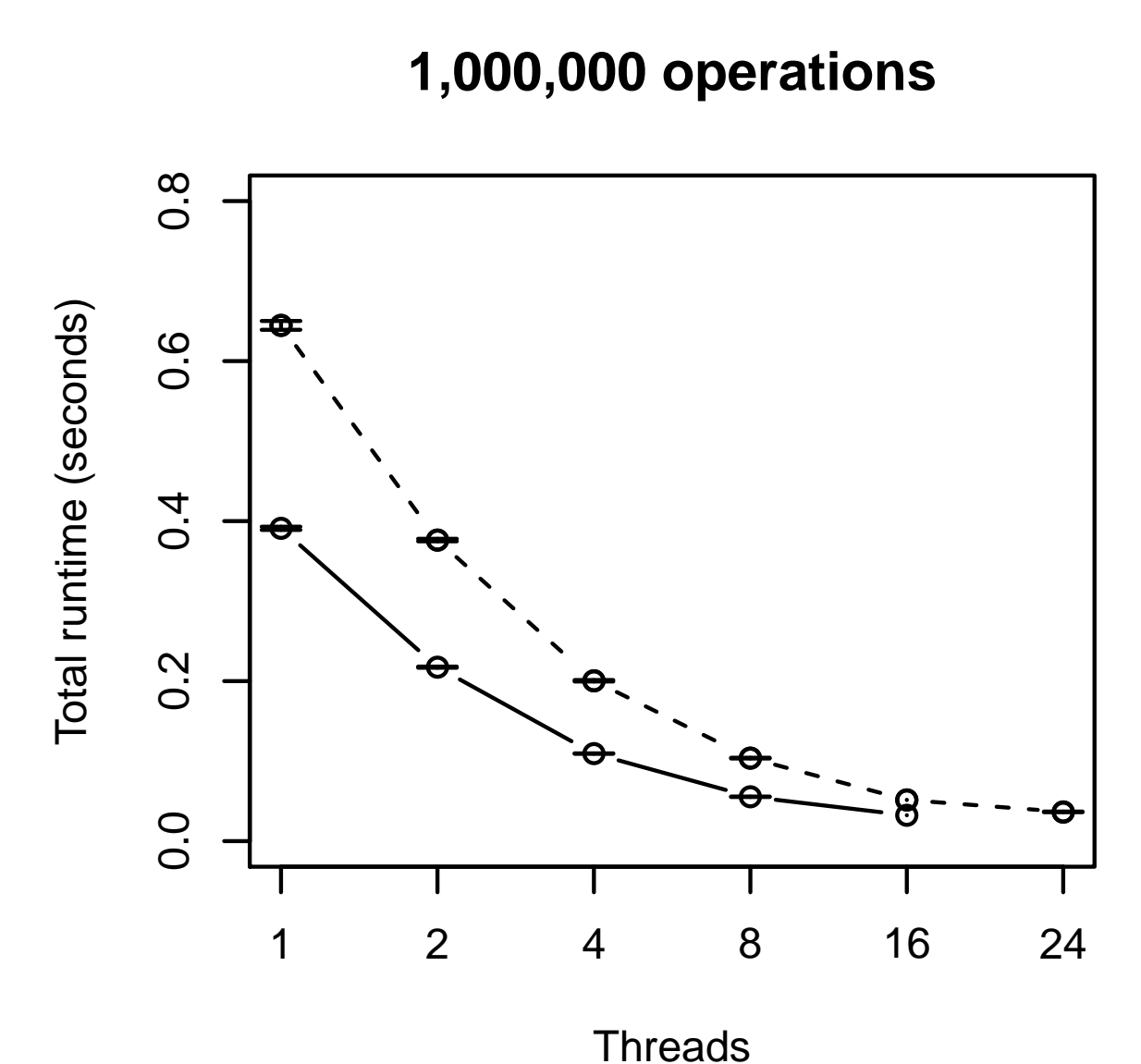
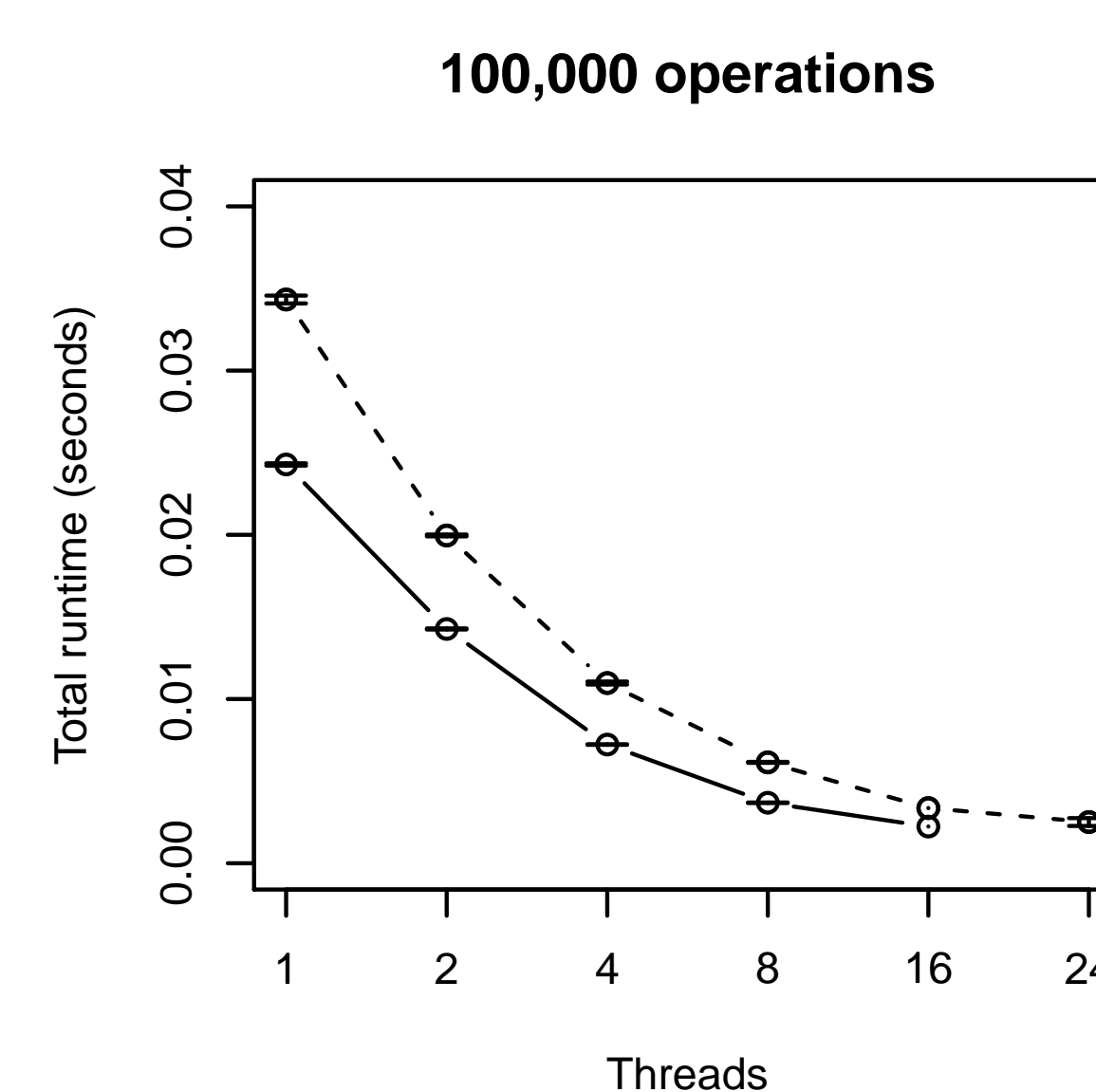
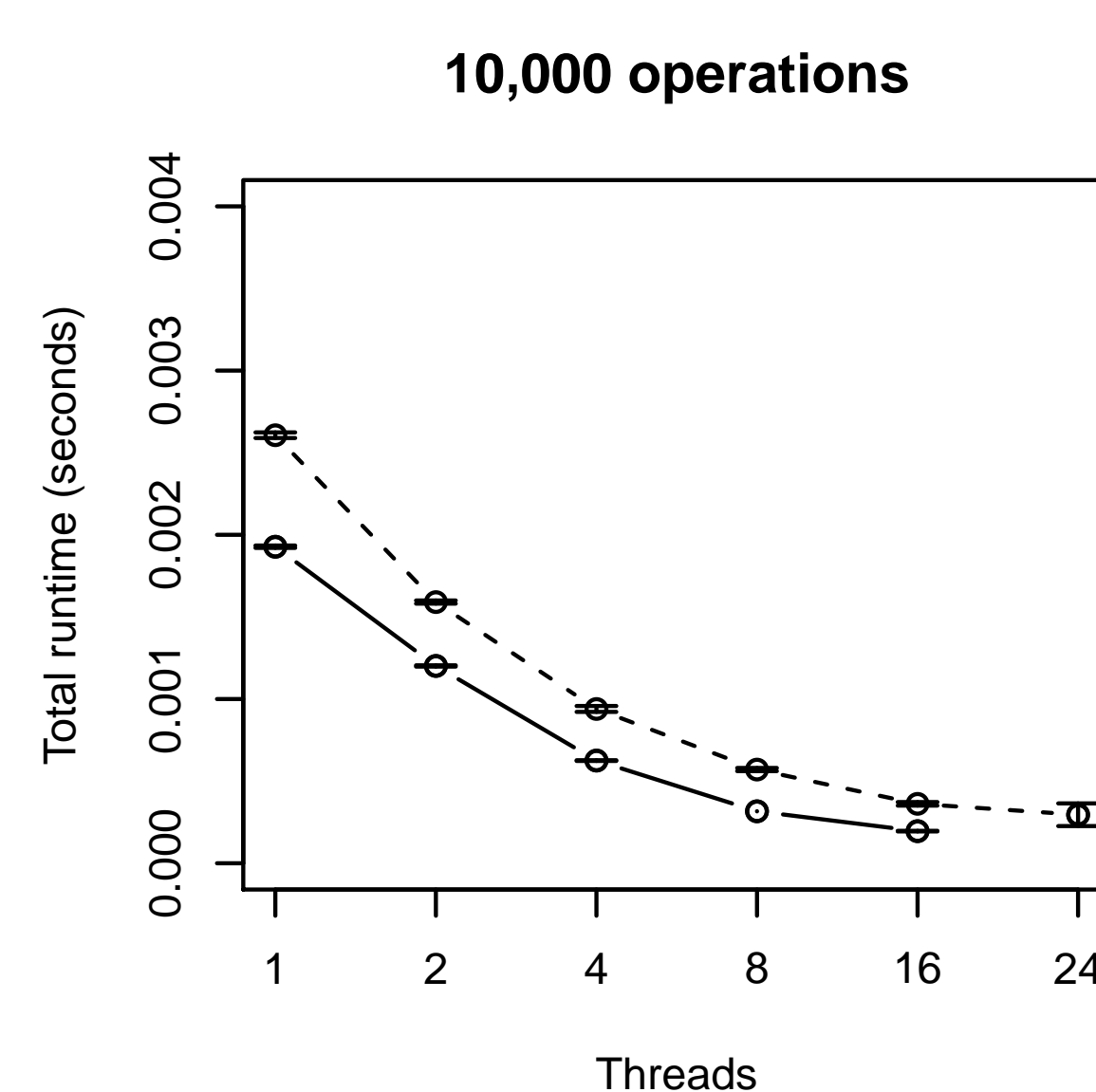
- ▶ Insert/remove: Find the relevant leaf and write with a CAS operation



- ▶ Rebalance: prevent changes to related nodes (in purple), replace the parent, and permit modification of grandparent

Evaluation

- ▶ Perform 10,000, 100,000, or 1,000,000 operations on tree of size 10,000, 100,000, or 1,000,000
- ▶ 20% of the operations are insert, 20% are remove, and the last 60% are search operations
- ▶ Uniformly distributed keys and values
- ▶ Measure runtime and speedups relative to single threaded left-leaning red-black tree
- ▶ Solid line is 2x 4 Core Intel Xeon, dashed line is 2x 16 core AMD Opteron



- ▶ Better speedup for large than small trees, due to:
 - ▶ Spatial locality more significant, lower node contention, lower relative overhead for leaf processing

Limitations

- ▶ Limited to 32 bit keys and values
- ▶ Operations not linearizable
- ▶ Requires 128 bit CAS operations

Related Work

- ▶ Ellen *et al.* A Lock-Free B+tree. In *PODC'10*.
- ▶ Braginsky *et al.* A Lock-Free B+tree. In *SPAA'12*.
- ▶ Bonnichsen *et al.* ELB-trees. In *MuCoCoS'13*.

Conclusion

- ▶ Introduced a scalable, efficient, and thread safe dictionary
- ▶ Comparable sequential performance to left-leaning red-black trees
- ▶ Highly scalable, especially for large data sets